Renarrating Linguistic Architecture: A Case Study

Vadim Zaytsev
Software Analysis & Transformation Team, Centrum Wiskunde & Informatica
Amsterdam, The Netherlands
vadim@grammarware.net

ABSTRACT

We study the use of megamodels (models of linguistic architecture) for presenting software language engineering scenarios. Megamodels and techniques similar to them are frequently found in situations when a linguistic architecture needs to be understood without the implicit knowledge that was originally present, and in situations when such knowledge needs to be propagated. In this paper we specifically address the possibility of using one megamodel to tell several related stories — that is, to renarrate it. Various renarrations can address different aspects of the megamodel, without cluttering the reader's view with irrelevant details. The renarration method is presented with the case study of a software language engineering technique of guided grammar convergence, and MegaL as a metamegamodel.

1. INTRODUCTION

The term "renarration" is used in natural language processing and database journalism to describe the process of converting a collection of facts into a story. Specific to renarration is the anticipation of conflicts: while generally the research on "views" assumes them to be consistent with one another modulo some hidden or rearranged details, it is normal and expected of several renarrations to deliver conflicted messages [1]. The same is often true for big megamodels.

The term "megamodelling" [2, 4] refers to the higher level of modelling that specifically addresses relationships between complex entities such as software languages and model transformations, aids in expressing software technologies and relating technological spaces [8]. Ad hoc megamodelling with throwaway notations or with variants of existing problem-specific notations is often used in situations when complicated setups need to be documented or explained, and has proven to be very useful [5, 10, 11], but also more systematic attempts to develop a unified megamodelling visual language for arguing about any kinds of linguistic architecture have been made [3] — one of such attempts is MegaL which will be used for this paper (Figure 1).

Megamodelling is a technique aimed at system comprehension, yet even relatively small megamodels describing one particular method, are often too big to be consumed by unattended readers. As a running example, consider Figure 2, depicting a reasonably detailed megamodel of the guided grammar convergence technique being developed by the author [12]. Every detail present on this model is relevant to the method and is possibly either a contribution or a limita-

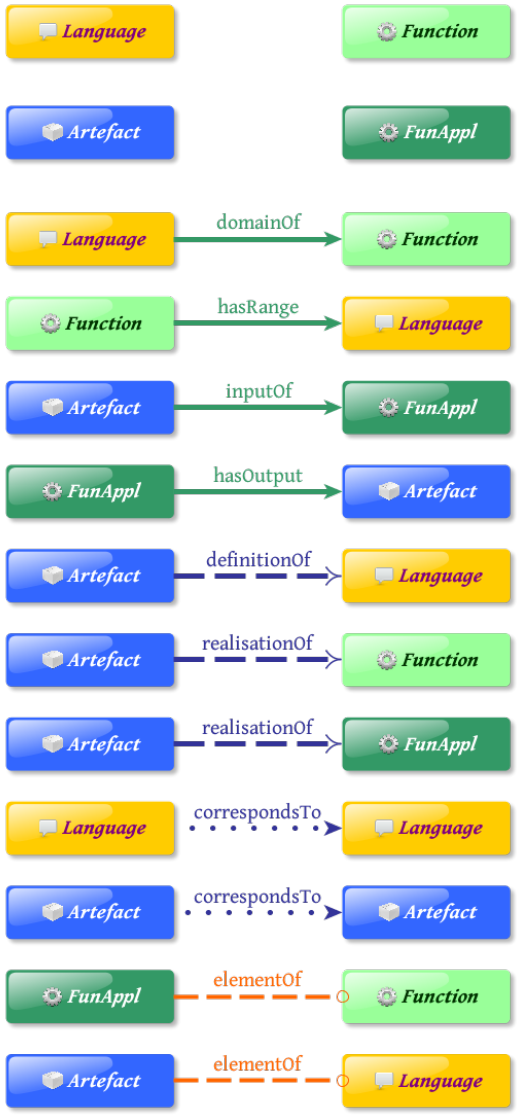

Figure 1: Core entities in MegaL models in this paper: artefacts, languages, functions and function applications, and possible relationships between them. Italicised labels denote variables, normal font labels always refer to concrete entities.

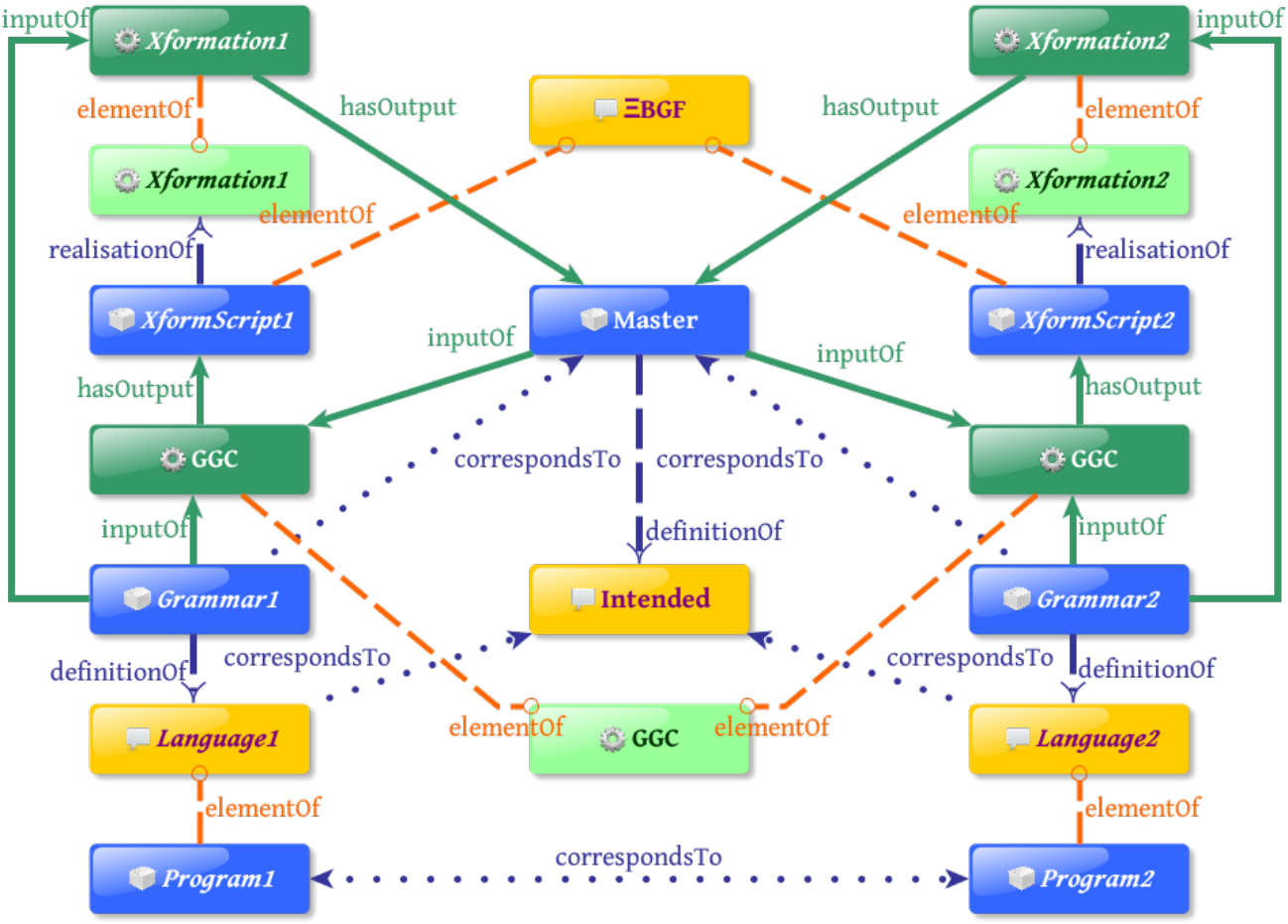

Figure 2: A minimal complete megamodel of the guided grammar convergence technique [12].

tion of it. This means that every element of the model must have a story attached to it in order for the megamodel to be useful. We argue that this megamodel can be made easily consumable by deriving several smaller megamodels from it and narrating them — this is the process that we will call "*renarration*", for the reasons explained above.

2. MEGAL

MegaL is a unified megamodelling language proposed by Favre, Lämmel and Varanovich in [3]. On the pages of this paper, we use a subset of the MegaL/yEd visual notation with entities of four types, which is explained in sufficient level of detail to not require acquaintance with the original paper. The entity types are distinguished by the colour (second shown is for grayscale) and an associated icon:

- *Language* (yellow/light gray, talk bubble icon): a software language in the broad sense of creating languages using metamodels [6].
- *Artefact* (blue/dark gray, box icon): a tangible software artefact—i.e., a file, a file fragment, a language definition, a language instance, a library etc.
- *Function* (light green/light gray, cogwheel): a function in the model transformation sense; for the sake of brevity, partially evaluated functions are also depicted as functions.
- *Function application* (dark green/dark gray, cogwheel): a concrete transformation, usually conforming to some function definition, but also having all arguments at its disposal.

Figure 1 lists all MegaL relationship types [3] that will be used within this paper:

- We say that *a language is a domain of a function* when the function operates on structured input that commits to the grammatical structure of the language [7].
- We say that *a function has a language as range* when the function produces structured content that commits to the structure of that language.
- *An artefact is an input of function application*, when this artefact (a file, a program, another tangible entity) is necessary for the function to run and serves to instantiate the function.

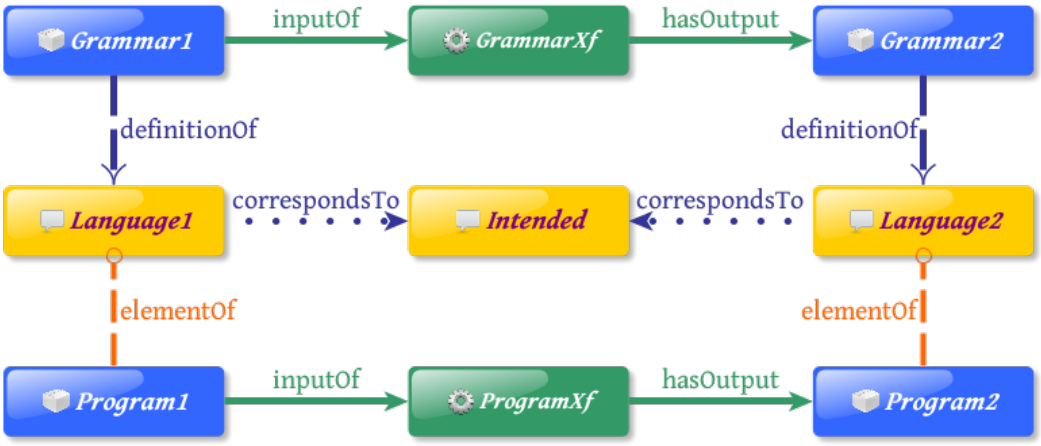

Figure 4: Guided grammar convergence motivation: the correspondence and transformability properties are articulated explicitly by the presence of the intended language and two transformation functions.

- Similarly, *function application has an artefact as an output*, when this artefact is expected to be generated when a valid input is provided.
- *An artefact is a definition of a language*, when it constitutes a specification of the structural definition of this language (e.g., a grammar, a metamodel, a schema).
- *An artefact is a realisation of a function or a function application*, when it represents a tangible serialisation of this function or a function application.
- *A language corresponds to a language*, when there is some (possibly unknown explicitly) correspondence relation between instances of the languages. This correspondence can be depicted as unidirectional or bidirectional, depending on the intended emphasis.
- *An artefact corresponds to an artefact*, when there is some (possibly unspecified) correspondence relation between the artefacts.
- *Function application is an element of a function*, when the function application truly conforms to the function definition and is supplied with appropriate parameters.

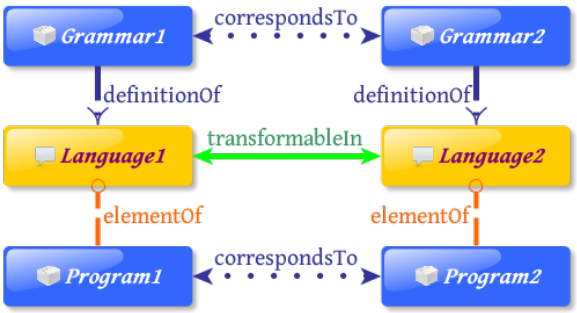

Figure 3: Guided grammar convergence motivation: we introduce a "transformableIn" relation type.

- *An artefact is an element of a language*, when it commits to its grammatical structure and can therefore be claimed to be an instance of that language.

3. RENARRATIONS

In the following subsections we present parts and slices of the megamodel from Figure 2 and narrate their stories. The analysis of these renarrations will follow in §4.

3.1 Motivation

The need for a guided grammar convergence technique is motivated by the need to establish and maintain executable relationships between software languages [12].

NARRATION 1. *Consider Figure 3: if we have any two grammars that exhibit some correspondence, each of those grammars defines its language, which is populated by appropriately conforming programs that also have correspondence across languages. We want these languages to be transformable in each other — that is, we assume the existence of appropriate transformation functions, but intentionally do not introduce them directly.*

NARRATION 2. *Consider Figure 4: if we have any two grammars, then the languages they define, should both have correspondence to some common intended language, in order for this technique to be applicable. If this is the case, we may want to obtain either a grammar transformation, which, applied to one of the given grammars, yields another one; or a program transformation that performs a similar task on the language instance level; or both.*

Narration 2 is more detailed than Narration 1, but does not introduce a new relation and can help establishing ground for further explanations of the intended language and the transformations. Both operate only on abstract uninstantiated entities: the fact that the intended language is fixed before the start of convergence, is irrelevant to establishing the motivation.

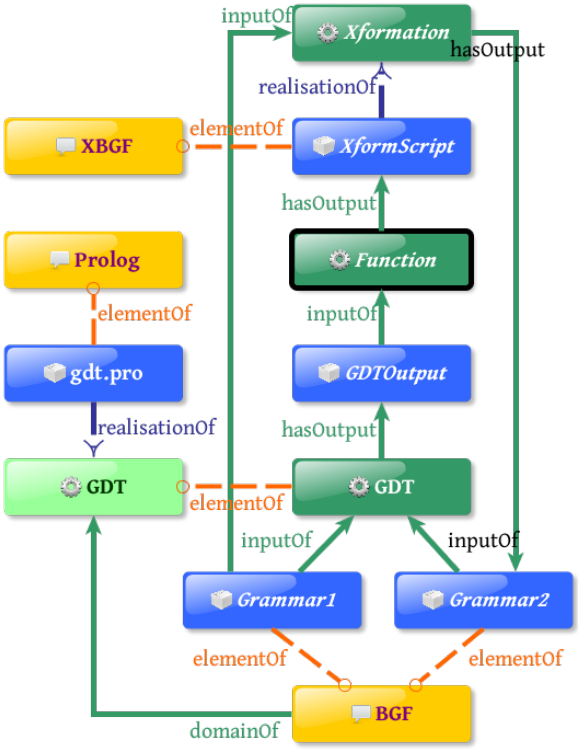

Figure 5: Contemporary grammar convergence relies on the output of the Grammar Diff Tool (GDT) being turned into the transformation script by an experienced grammar engineer.

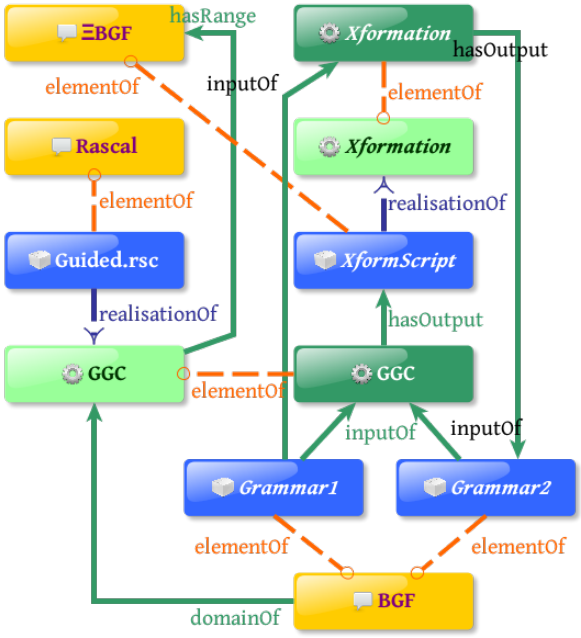

Figure 6: Guided grammar convergence (GCC) produces a bidirectional transformation script automatically.

3.2 Technique

If we assume that the motivation is clear and established, we can focus on the technical contribution of the guided grammar convergence method:

Narration 3. *In the original grammar convergence approach (Figure 5, see [9]) we have a Grammar Diff Tool to report differences between any two given grammars. Consuming its output repeatedly and turning it into a transformation script that resolves any spotted mismatches, is a function (marked by a bold black border) performed by a human stakeholder — a grammar engineer who is familiar with the transformation operator suite. In the case of guided grammar convergence (Figure 6), the transformations are directly inferred by an automated tool that consumes both input grammars.*

Narration 4. *As can be seen on Figures 5 and 6, the guided grammar convergence method outputs the transformation script directly without relying on a human grammar engineer for support (its function marked by a bold black border). Besides that, it also uses bidirectional ΞBGF as a base language instead of unidirectional XBGF, so all inferred transformations are bidirectional by construction. The prototyped algorithm of guided grammar convergence is realised in Rascal, which makes it easier to reuse the technology from within the Eclipse IDE.*

Both narrations use *two* megamodels as a base for their story: the second one is an enriched slice of the original megamodel from Figure 2, while the first one is a revised adjustment, since it is needed to explain a different technique (the original method on which guided convergence is based). Narration 4 can be used as a plain extension of Narration 3, but we intentionally phrase it differently to raise the level of abstraction for the same megamodels.

3.3 Programmable transformations

As it has been explained in the motivation section, by language correspondence or by its transformability we assume the existence of transformation functions. Let us consider them closer and explain some details not present in or not apparent from the original megamodel.

Narration 5. *A bidirectional grammar transformation can be seen on Figure 7, where the composition is shown with boxing and not with connectors only for the sake of visual clarity. Any instance of the ΞBGF language implements a bidirectional transformation, which has two possible applications: forward and reverse. These two applications agree to the extent that their inputs and outputs are grammars specified in the same BNF-like Grammar Formalism (BGF) language.*

Narration 6. *A bidirectional grammar transformation can be seen on Figure 8. Any instance of the ΞBGF language implements a bidirectional transformation, which has four possible applications: forward and reverse, language level and instance level. Combinations of these form pairs: the two applications that work on the grammar level, agree on*

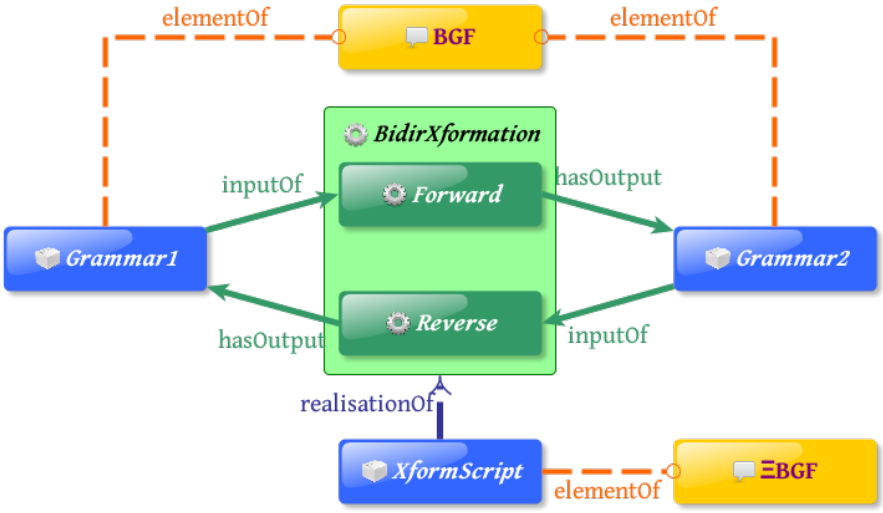

Figure 7: Bidirectional grammar transformations.

their inputs and outputs; the two forward applications also agree on producing a program that commits to the grammatical structure of the produced grammar.

These two narrations are substantially different, and the choice depends not only on the audience, but also on the factual side: which scenario was technically implemented? (In our case, Narration 5 explains the current state and Narration 6 relates to future work [13]).

3.4 Conclusion

After the fragments of Figure 1 are explained individually, it becomes much easier to consume the whole megamodel: we start with two grammars, assume the intended language, substantiate the assumption by composing a master grammar that defines it, and run the guided grammar convergence algorithm in order to infer bidirectional grammar transformation steps.

4. SUMMARY

This case study was meant to demonstrate some possible uses for renarrating megamodels or similar high level models of linguistic architecture of a software system. Renarration is a method from which we can benefit not only for giving presentations and lectures, but also for writing more easily understandable papers and for packaging useful auxiliary material to accompany those papers online.

The claim that we need more than simple grouping and slicing, was demonstrated by the following cases.

- In Narration 1 (Figure 3) we introduce a new relationship kind (transformableIn).
- In Narration 2 (Figure 4) we narrate a concrete entity (the intended language) as a variable.
- In Narrations 3 and 4 we used two megamodels (Figures 5 and 6) for one story.
- In Narrations 3 and 4 we used a severely adjusted megamodel slice (Figure 5) to show the difference with the previously published work.
- Figure 5 used in Narrations 3 and 4, intentionally conflicts the baseline megamodel to unveil parts of its evolution.
- In Narration 5 (Figure 7) the megamodel was sliced and refined to expose some implementation details.
- In Narration 6 (Figure 8) the megamodel was sliced and refined to sketch future implementation plans.
- In Narrations 5 and 6 (Figures 7 and 8) a new entity notation was introduced for encapsulating several possible function applications in one function..

Currently ongoing work on renarrating linguistic architecture focuses on the following items:

- Composition of the disciplined megamodel transformation operator suite appropriate for renarration, so that all renarrations shown above can be represented systematically as transformation scripts.
- Partial automation of the activities of renarration: smart slicing, shadowing, verification; tool support thereof.
- Technical support for renarrations of megamodels as clickable animated visualisations.
- Modelling multi-megamodel scenarios such as renarration, to uncover the big picture of their results at a glance.
- Investigating possibilities and prospects of the application of renarration to software engineering in general, in particular to comprehension and visualisation techniques other than megamodelling.

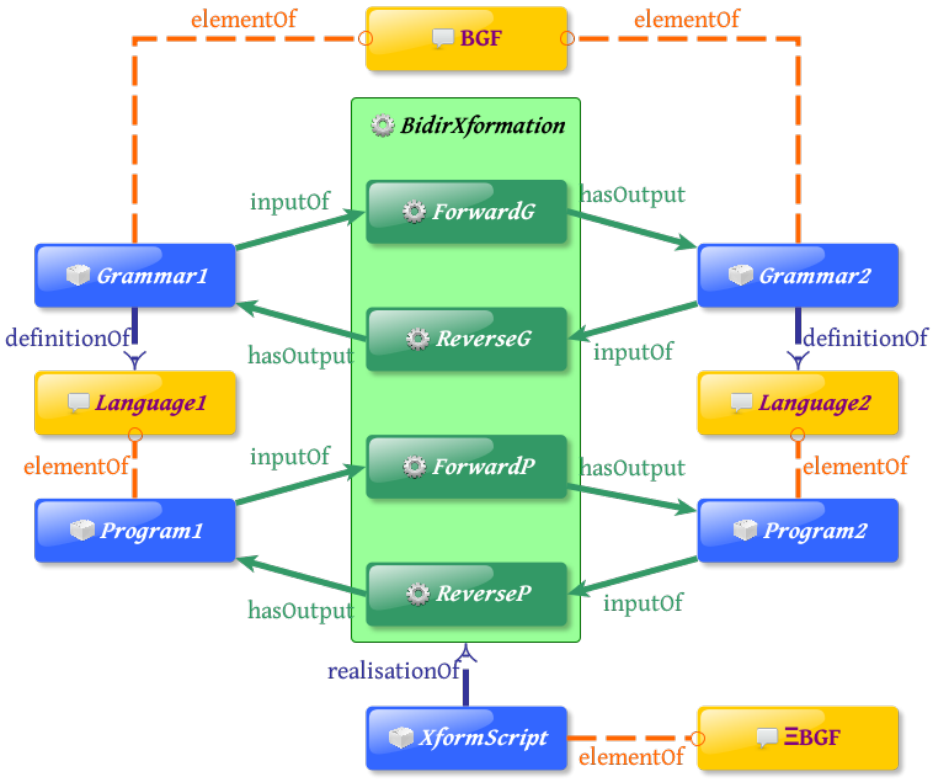

Figure 8: Bidirectional grammar transformations and coupled bidirectional instance transformations.

Acknowledgement

The author would like to express his gratitude to T. B. Dinesh who has introduced him to the notion of renarration at the PEM Colloquium at CWI, as well as to Jean-Marie Favre and Ralf Lämmel for fruitful discussions on megamodelling.

5. REFERENCES

[1] M. Baker and A. Chesterman. Ethics of renarration. *Cultus*, 1(1):10–33, 2008. Mona Baker is interviewed by Andrew Chesterman.

[2] J. Bézivin, F. Jouault, and P. Valduriez. On the Need for Megamodels. *OOPSLA & GPCE, Workshop on best MDSD practices*, 2004.

[3] J.-M. Favre, R. Lämmel, and A. Varanovich. Modeling the Linguistic Architecture of Software Products. In *Proceedings of MODELS 2012*, LNCS. Springer, 2012. 17 pages. To appear.

[4] J.-M. Favre and T. NGuyen. Towards a Megamodel to Model Software Evolution through Transformations. *Electronic Notes in Theoretical Computer Science, Proceedings of the SETra Workshop*, 127(3), 2004.

[5] R. Hilliard, I. Malavolta, H. Muccini, and P. Pelliccione. Realizing Architecture Frameworks Through Megamodelling Techniques. In *Proceedings of ASE'10*, pages 305–308, New York, 2010. ACM.

[6] A. Kleppe. *Software Language Engineering: Creating Domain-Specific Languages Using Metamodels*. Addison-Wesley Professional, 2008.

[7] P. Klint, R. Lämmel, and C. Verhoef. Toward an Engineering Discipline for Grammarware. *ACM TOSEM*, 14(3):331–380, 2005.

[8] I. Kurtev, J. Bézivin, and M. Akşit. Technological Spaces: an Initial Appraisal. In *Proceedings of CoopIS, DOA'2002, Industrial track*, 2002.

[9] R. Lämmel and V. Zaytsev. An Introduction to Grammar Convergence. In M. Leuschel and H. Wehrheim, editors, *Proceedings of iFM 2009*, volume 5423 of *LNCS*, pages 246–260. Springer-Verlag, February 2009.

[10] B. Meyers and H. Vangheluwe. A Framework for Evolution of Modelling Languages. *Science of Computer Programming*, 76(12):1223 – 1246, 2011. Special Issue on Software Evolution, Adaptability and Variability.

[11] V. Zaytsev. *Recovery, Convergence and Documentation of Languages*. PhD thesis, Vrije Universiteit, Amsterdam, The Netherlands, 2010.

[12] V. Zaytsev. Guided Grammar Convergence. Full Case Study Report. Technical report, CWI, 2012. Available at http://grammarware.net/writes/#Guided2012.

[13] V. Zaytsev, R. Lämmel, T. van der Storm, L. Renggli, and G. Wachsmuth. Software Language Processing Suite[1], 2008–2012. http://grammarware.github.com.

[1]The authors are given according to the statistics at http://github.com/grammarware/slps/graphs/contributors.